Storage of Simulation and Entities History in discrete models

De Giusti Marisa Raquel[*], Lira Ariel Jorge[†], Villarreal, Gonzalo Luján[**].-

[*] PrEBi UNLP and Comisión de Investigaciones Científicas (CIC) de la Provincia de Buenos Aires. La Plata, Argentina
[†] PrEBi UNLP. La Plata, Argentina.
[**] PrEBi UNLP and Consejo Nacional de Investigaciones Científicas y Técnicas (CONICET). La Plata, Argentina

Index Terms: Models, Simulation, Persistence, Databases

Abstract

Simulation is the process of executing a **model**, that is a representation of a system with enough detail to describe it but not too excessive. This model has a set of entities an internal state, a set of input variable that can be controlled and others that cannot, a list of process that bind these input variables with the entities and one or more output values, which result from the excecution of the processes.

Running a model is totally useless if it can not be analized, which means to study all interactions among input variables, model entities and their weight in the values of the output variables. In this work we consider Discrete Event Simulation, which means that the status of the system variables being simulated change in a countable set of instants, finite or countable infinite.

Simulation programming languages provide a big range of tools for analysis of the results, for generation and execution of experiments and to perform complex analysis (such as Analysis of Variance). This is usually enough for common analysis, but many times more detailed information is required.

In many circumstances it is desirable to have all run of simulations stored in order to make further analysis or comparissons between simulations, many time after they have been run. Most simulation environments provide reports and logs (journals) and permit to save them in text or formated files, which include all results and some aspects of the run itself.

In this work, we propose to store not only all simulation results, but all simulation history. This implies to store all permanent entities of the simulation, and all changes they have undergone along the simulation times. But it not only limits to permanent entities, since we also store the temporary ones, which are created and destroyed anytime in the simulation and which existence is subject to the execution of the simulation and the state of all model variables.

This development takes the same syntax of GPSS language and the way it handles entities, to develop a simple interpreter and a tool that considers a subset of GPSS entities and permits to execute simulations with them. Besides, this tool permits to search and select entities for each simulation and displays their evolution along the simulation

[*],[†],[**]{marisa.degiusti, alira, gonetil }@sedici.unlp.edu.ar

1. Introduction.

Block programming languages offer an important abstraction level that allows the programmer to map objects from the real system to entities of the simulation in an almost transparent way, providing implicitly with each object a set of functions and facilities, which lets the programmer focus on designing the model and forget about programing of implementation details. This is particulary useful when looking for quick solutions, since time savings achieved can be very significant and this the required cost for modeling, simulating and experimenting with systems goes considerably down.

The real aim of any simulation is not to run (execute) a system in a computer but to gather as much information as possible for making all the analysis we can imagine. Simulation languages usually collect all posible data while the along the simulation run, and most of them offer tools for displaying this information, using simple text reports and probably statistics graphs, tables and functions. GPSS programming language, which this work is based on, is not an exception.

Besides all information given once the simulation has ended, it is very important to know how our mathematical model has reach this final state and why it has happened this way. This kind of analysis is far too complicated in discrete models executed in GPSS-like languages, specially average or large models with many entities, transactions and processes inside. Thousands of transactions might been created, moved and destroyed, and it is really hard to track them back individually.

In this work we present an incipient open simulation framework that allows to store in a database all simulation objects in order to know how they have evolved along the simulation time and what has happened in each instant. This allows the analyst, for example, to pick any entity and know all changes it has suffered in all simulation times, which entities it has interacted with and what was its time life inside the model. In addition, they may select a range of time and see what entities existed at that moment. Or we can go even further, and consider running and persisting many simulations with different parameters and then compare information from different simulations runs.

As we have mentioned, it is an open and incipient framework. It is open because it has been created according to GPL licence, and incipient because up to now it implements all basic model which permits to run simple simulations but to extend it including all we want.

2. The model.

For the development of this framwork a subset of GPSS entities have been selected, which allow us to execute simple simulations but that require the execution model to be complete enough. Among these entities we have Transaction, Facility and, clearly, Simulation.

Simulation entity is in charge of the execution of commands and the generation of transactions; it is also task of this entity to invoke the transaction scheduler which must decide which transaction have to be the next active transaction, and to manage the simulation clock, detecting when it must update and making all required tasks in each clock change. This implied the addition of Transaction Scheduler and System Clock entities inside the model. It has also been added an spare entity which deals with references to entities resolution, either by name or identifier; is entity is able to locate any entity at any time anywhere in the simulation.

All entities form an horizontal composition cyclic graph, being the entity Simulation as the root of this graph, from which all other nodes can be visited (Fig. 1). The entity Transaction Scheduler is extremely complex since it must interact, at first, with the two main chains of a simulation: current events chain and

future events chain. But this entity must also be able to access transaction held by other chains in each existing entity every time is needed; for example, if the transaction is trying to release the facility, the transaction scheduler must select which transaction will be the following owner of the facility being released, meaning that it must analyze the current events chain, the delay chain of the facility, the preempt chain and the interrupt chain.

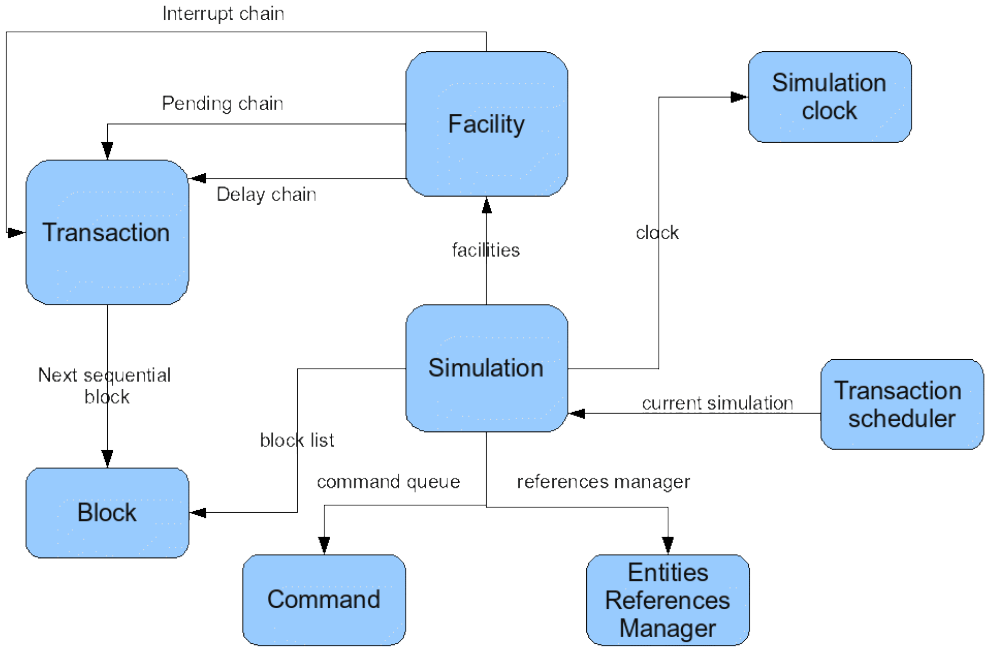

Figure 1: Simulation graph

The movement among chains is also a difficult task, since in implies not only the exchange of hundreds or thousands transactions from chain to chain, but it must also select transactions to exchange and choose the right moment to make this operation (change in the system clock, delay condition testing, and others).

The decision of including in this first version of the framework the entity Facility was not taken randomly; this entity involves a wide set of functions and entities, and has an intrincate method for selecting transactions according to the internal state of each one of its chains and the way it is being accessed.

3. Simulation persistence.

In order to store simulation in the disk, a relational database engine (MySQL) together with a java ORM (JPOX) has been used. As mentioned above, we are dealing with discrete simulations, hence we have a countable set of *simulation instants* in which many events have occurred and entities have been altered. So, we have consider to take a picture of the whole simulation, including all entities with their internal state, and to store this picture exactly as it is. The storage process happens every time the simulation clock is updated, just before this events actually occurs, in order to persist the simulation with the state before the clock changes.

Store a simulation in a determined time t implies to store all entities that take part in this simulation at this time t. In order to collect all entities, the composition graph is walked along, starting in the node representing the simulation entity and moving forward all simulation chains, entities list and chains of every entity. Since blocks are also entities, they must be persisted too.

Everytime the graph is walked along, all persitible information is collected and, once it is finished, the simulation clock is updated and the execution goes on. Then, ideally, all this data should be persisted to disk and the system clock updated in order to go on with the run. The problem is that writting to the disk is too slow compared to main memory and processor speed, and we can not let the simulation stop until objects are stored.

Our solution to this problem consist of having a ready-to-store queue, and a low priority thread in charge of the persistence (Fig. 2). All data ready to persist is queued and remains in there until it is actually persisted, and the simulation continues running without waiting until data is written to disk. To persist objects, we have include a single thread, which runs together with the rest of the simulation. This thread picks up simulations ready from the persist queue and deals with DAO object and all database stuff (connections, queries, transactions and so on). It is important to remark that the extra thread has a lower priority than the simulation run; we have designed it this way because we do not want the simulation to be delayed for anything.

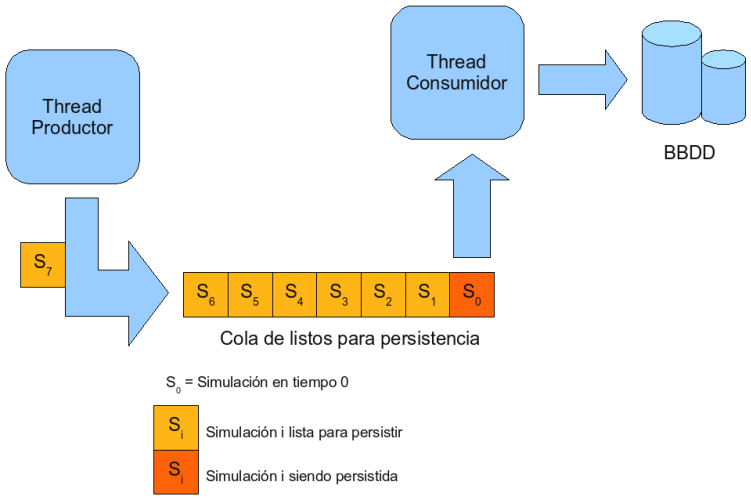

Figure 2: Producer/Consumer scheme

This solutions is good for having results immediately when the simulation ends while the persistence to the database is being made in background. The user may start analyzing data or adapting the model while the simulation keeps being stored in background, improving user experience and efficiency.

3.1 Data collection.

The whole application is written in Java, which means it uses all well known Object Oriented concepts (inheritance, hierarchy, OID, etc). When saving this objects to the database, we must make sure that they are not updated since we would loss the state of the objects before this update. One solution would be to have a version scheme where only new or changed data are stored; the problem of the versioning solution is that it would be really difficult to retrieve a simulation in any system clock. It would be efficient in disk, slow in data retrieve though.

Since hard disks are really huge these days, and computers have lots of memmory, it is not expensive to duplicate data even if it has not changed. With this idea in mind, our solution has been both simple and efficient: the whole simulation graph is cloned in every change of the system clock, copying every node that takes parts of the simulation (transactions, facilities, blocks, chains, and everything else). Once the clonning process has ended, se clonned simulation is queued to store and, as said above, the simulation can move on.

4. Data recovery.

The way data are stored permits to retrieve simulations very easily; in the database there are many copies of every simulation run, where each copy will differs in, at least, the state of the simulation clock. Hence, to recover a simulation in a specific time t is as simple as retrieve the Simulation object and again, start a chain effect retrieving all objects reachable from it.

But being able to retrieve a simulation in a time t is no the only advantage of this tool. All entities, both permanent and temporary, have an internal identifier, which never changes along the simulation. This is very useful to write new analysis tools that pick an specific entity and all copies generated before it; with all this information, this tool could show exactly how the entity changed, or which transaction were accessing it or were in its chains, or any other data that determine its internal state. This tool could even go beyond this ideas: we could have stored many simulation executions for the same model but with different parameters, and then we could retrieve any particular entity with all its states for all simulations. This way, we can not only see how this entity has changed, but to compare this changes with all instances of the same entity in different simulations.

6. Conclusion.

The framework is still being developed, incorporating new entities and blocks and improving the already developed ones. Besides the completion of GPSS model, this system is a great opportunity to every programmer who wants to add his own analysis tool, since all data of every simulation is ready to access and use; the fact that the framework is opensource permits programmers to understand how data is organized, so they can efficiently retrieve it and show it the way they want.

This kind of tool is also very useful to help students during the learning process of GPSS (and other simulation tools); we have studied GPSS model and grouped a subset of entities to develop a base model, which has to be extended but is full enough to start playing with simple yet powerful simulations. Studens may already open any entity and see how it works below, how things are done and how it could be improved. Students and any programmer can even create their own entities beyond the ones defined in GPSS; there are no limits at all.

References.

[AND99] Foundations of multithreaded, parallel and distributed programming. Gregory R. Andrews. Addison Wesley. 1999

[DUN06] Simulación y Análisis de sistemas con ProModel. Eduardo García Dunna. Heriberto García Reyes. Leopoldo E. Cárdenas Barrón. Pearson. 2006.

[JPO08] Java Persistent Objects http://www.jpox.org/

[LAN85] Teoría de los sistemas de información. Langefors, Börje. 2a. ed. (1985)

[KIM88]Schema versions and DAG rearrangement views in object-oriented databases (Technical report. University of Texas at Austin. Dept. of Computer Sciences). Hyoung Joo Kim. University of Texas at Austin, Dept. of Computer Sciences. 1988

[KIM90] Introduction to Object-Oriented Databases. Won Kim. The MIT Press. 1990.

[MIN05] Minuteman Software http://www.minutemansoftware.com

[SIM08]. Simulation and Model-Based Design http://www.mathworks.com/products/simulink/

[WIL66] Computer simulation techniques. Wiley; 1st corr. printing edition (1966)